Malware-Analysis/Cobalt Strike/Indirect Syscalls.md at main · dodo-sec/Malware-Analysis

By dodo-sec

Archived: 2026-04-05 20:55:48 UTC

An analysis of syscall usage in Cobalt Strike Beacons

Thanks to the suggestion of my good friend Nat (0xDISREL), I spent the last week digging into a Cobalt Strike beacon made with the latest leaked builder. His idea was to analyze and understand how CS approached syscalls.

Sample

This analysis was conducted in an x64 bit payload with the hash `020b20098f808301cad6025fe7e2f93fa9f3d0cc5d3d0190f27cf0cd374bcf04`, generated by the recently leaked 4.8 version of Cobalt Strike. It's publicly available for download in unpacme. I will not go over unpacking the sample for the sake of brevity, but doing so is pretty straightforward and shouldn't present any problems.

A quick refresher

Before we get to the actual reversing, let's get a quick refresher on what system calls look like under Windows.

According to calling convention, arguments are setup in the appropriate registers before the instruction `SYSCALL` is executed, handling execution to the Kernel. One of such arguments is the code for the system call (in the picture

above, it's passed via the `eax` register). These system calls reside in ntdll and provide evasion benefits by allowing you to avoid calling APIs that are likely hooked by AV/EDR.

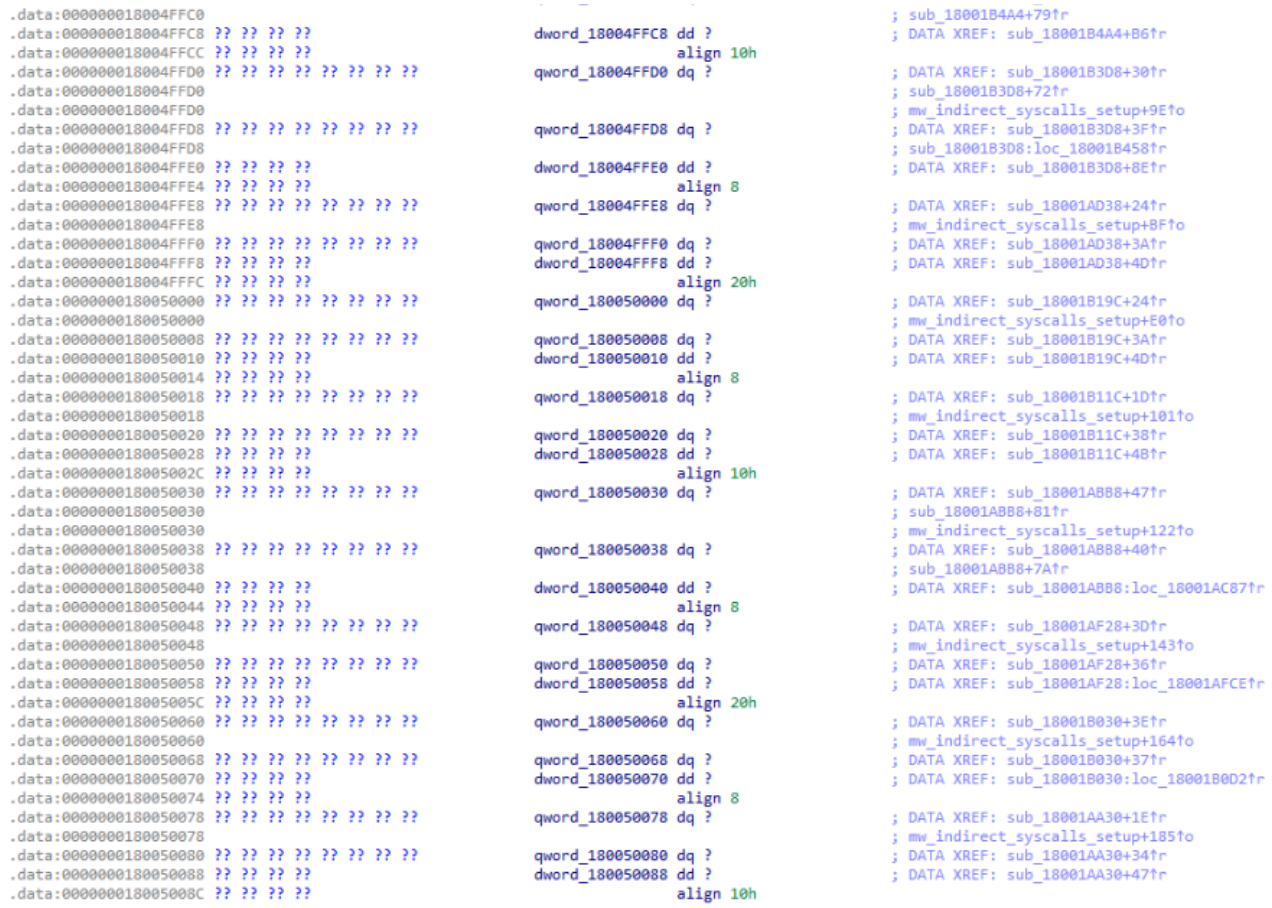
How Cobalt Strike does it

During the first steps of analysis of the unpacked payload we'll come across references to qwords and calls to registers.

```
.text:0000000180018421
.text:0000000180018421 48 89 54 24 58
.text:0000000180018426 48 89 4C 24 20
.text:000000018001842B FF 15 7F 4E 01 00
.text:0000000180018431 83 3D 34 08 03 00 01
.text:0000000180018438 44 8B CB
.text:000000018001843B 4C 8D 44 24 58
.text:0000000180018440 4C 8B D0
.text:0000000180018443 48 8D 54 24 20
.text:0000000180018448 75 0E
.text:000000018001844A 48 8B 05 7F 4B 03 00
.text:0000000180018451 49 8B CA
.text:0000000180018454 FF D0
.text:0000000180018456 EB 22
.text:0000000180018458
.text:0000000180018458
.text:0000000180018458
.text:0000000180018458 48 8B 0D 79 4B 03 00
.text:000000018001845F 48 89 0D FA 07 03 00
.text:0000000180018466 8B 0D 74 4B 03 00
.text:000000018001846C 89 0D F6 07 03 00
.text:0000000180018472 48 8B C8
.text:0000000180018475 E8 0C 1C 00 00
.text:000000018001847A
.text:000000018001847A
.text:000000018001847A 85 C0
.text:000000018001847C 75 07
.text:000000018001847E B8 01 00 00 00
.text:0000000180018483 EB 0F
.text:0000000180018485
.text:0000000180018485
.text:0000000180018485
.text:0000000180018485
.text:0000000180018485
.text:0000000180018485
.text:0000000180018485 44 8B C3
.text:0000000180018488 48 8B D7
.text:000000018001848B 48 8B CE
.text:000000018001848E FF 15 8C 4D 01 00
.text:0000000180018494
.text:0000000180018494
.text:0000000180018494 48 8B 5C 24 40
.text:0000000180018499 48 8B 74 24 48
.text:000000018001849E 48 83 C4 30
.text:00000001800184A2 5F
.text:00000001800184A3 C3
.text:00000001800184A3
```

```
loc_180018421:                          ; CODE XREF: sub_1800183D8+38↑j
                mov     [rsp+38h+arg_18], rdx
                mov     [rsp+38h+var_18], rcx
                call    cs:GetCurrentProcess
                cmp     cs:dword_18004BC6C, 1
                mov     r9d, ebx
                lea     r8, [rsp+38h+arg_18]
                mov     r10, rax
                lea     rdx, [rsp+38h+var_18]
                jnz     short loc_180018458
                mov     rax, cs:qword_18004FFD0
                mov     rcx, r10
                call    rax ; qword_18004FFD0
                jmp     short loc_18001847A
; ---------------------------------------------------------------------------

loc_180018458:                          ; CODE XREF: sub_1800183D8+70↑j
                mov     rcx, cs:qword_18004FFD8
                mov     cs:qword_18004BC60, rcx
                mov     ecx, cs:dword_18004FFE0
                mov     cs:dword_18004BC68, ecx
                mov     rcx, rax
                call    sub_18001D086

loc_18001847A:                          ; CODE XREF: sub_1800183D8+7E↑j
                test    eax, eax
                jnz     short loc_180018485
                mov     eax, 1
                jmp     short loc_180018494
; ---------------------------------------------------------------------------

loc_180018485:                          ; CODE XREF: sub_1800183D8+1F↑j
                                        ; sub_1800183D8+3D↑j
                                        ; sub_1800183D8+47↑j
                                        ; sub_1800183D8+A4↑j
                mov     r8d, ebx        ; dwFreeType
                mov     rdx, rdi        ; dwSize
                mov     rcx, rsi        ; lpAddress
                call    cs:VirtualFree

loc_180018494:                          ; CODE XREF: sub_1800183D8+AB↑j
                mov     rbx, [rsp+38h+arg_0]
                mov     rsi, [rsp+38h+arg_8]
                add     rsp, 30h
                pop     rdi
                retn
sub_1800183D8   endp
```

Inspecting said qwords will lead us to the `.data` section, where they don't hold any values (yet).



Inspecting other references to these addresses will land us in a function that looks a lot like an import by hash routine - there are repeated calls to the same function, each time passing a different hexadecimal value and a `.data` section address among its arguments.

```
.text:0000000180016B0
.text:0000000180016B0 48 89 5C 24 10                    mov     [rsp-8+arg_8], rbx
.text:0000000180016B5 48 89 7C 24 18                    mov     [rsp-8+arg_10], rdi
.text:0000000180016BA 55                                push    rbp
.text:0000000180016BB 48 8D AC 24 90 E1 FF FF           lea     rbp, [rsp-1E70h]
.text:0000000180016C3 B8 70 1F 00 00                    mov     eax, 1F70h
.text:0000000180016C8 E8 33 33 00 00                    call    __alloca_probe
.text:0000000180016CD 48 2B E0                          sub     rsp, rax
.text:0000000180016D0 83 64 24 30 00                    and     [rsp+1F70h+var_1F40], 0
.text:0000000180016D5 44 8D 40 CC                       lea     r8d, [rax-34h]  ; Size
.text:0000000180016D9 48 8D 4C 24 34                    lea     rcx, [rsp+1F70h+var_1F3C] ; void *
.text:0000000180016DE 33 D2                             xor     edx, edx        ; Val
.text:0000000180016E0 E8 8B 1D 00 00                    call    memset
.text:0000000180016E5 48 83 A5 80 1E 00 00 00           and     [rbp+1E70h+arg_0], 0
.text:0000000180016ED BF F4 01 00 00                    mov     edi, 1F4h
.text:0000000180016F2 4C 8D 85 80 1E 00 00              lea     r8, [rbp+1E70h+arg_0]
.text:0000000180016F9 48 8D 4C 24 30                    lea     rcx, [rsp+1F70h+var_1F40] ; void *
.text:0000000180016FE 8B D7                             mov     edx, edi
.text:0000000180B700 E8 EF F0 FF FF                     call    sub_18001A7F4
.text:0000000180B705 48 8B 9D 80 1E 00 00               mov     rbx, [rbp+1E70h+arg_0]
.text:0000000180B70C 48 8D 05 8D 48 03 00               lea     rax, qword_18004FFA0
.text:0000000180B713 48 8D 4C 24 30                     lea     rcx, [rsp+1F70h+var_1F40]
.text:0000000180B718 4C 8B C3                           mov     r8, rbx
.text:0000000180B71B 41 B9 69 7A 2B B1                  mov     r9d, 0B12B7A69h
.text:0000000180B721 8B D7                              mov     edx, edi
.text:0000000180B723 48 89 44 24 20                     mov     [rsp+1F70h+var_1F50], rax
.text:0000000180B728 E8 0F F0 FF FF                     call    sub_18001A73C
.text:0000000180B72D 48 8D 05 84 48 03 00               lea     rax, qword_18004FF88
.text:0000000180B734 48 8D 4C 24 30                     lea     rcx, [rsp+1F70h+var_1F40]
.text:0000000180B739 41 B9 8B CF 08 C5                  mov     r9d, 0C508CF8Bh
.text:0000000180B73F 4C 8B C3                           mov     r8, rbx
.text:0000000180B742 8B D7                              mov     edx, edi
.text:0000000180B744 48 89 44 24 20                     mov     [rsp+1F70h+var_1F50], rax
.text:0000000180B749 E8 EE EF FF FF                     call    sub_18001A73C
.text:0000000180B74E 48 8D 05 7B 48 03 00               lea     rax, qword_18004FFD0
.text:0000000180B755 48 8D 4C 24 30                     lea     rcx, [rsp+1F70h+var_1F40]
.text:0000000180B75A 41 B9 23 21 AF 35                  mov     r9d, 35AF2123h
.text:0000000180B760 4C 8B C3                           mov     r8, rbx
.text:0000000180B763 8B D7                              mov     edx, edi
.text:0000000180B765 48 89 44 24 20                     mov     [rsp+1F70h+var_1F50], rax
.text:0000000180B76A E8 CD EF FF FF                     call    sub_18001A73C
.text:0000000180B76F 48 8D 05 72 48 03 00               lea     rax, qword_18004FFE8
```

Case closed then, the empty qwords would receive pointers to the resolved API functions, right? All that's left is to identify the hashing algorithm and start renaming things? Well, not quite. This write-up is not called "analyzing import by hash", after all.

Let's take a look at the function that's called before all the hashes start showing up. I've named it `mw_prepare_indirect_syscalls`.

```
.text:000000018001A81A 4C 8B E9                                mov     r13, rcx
.text:000000018001A81D 41 B8 20 20 20 20                       mov     r8d, 20202020h
.text:000000018001A823 4C 8B 48 60                             mov     r9, [rax+TEB64.ProcessEnvironmentBlock]
.text:000000018001A827 4D 8B 51 18                             mov     r10, [r9+PEB64.Ldr]
.text:000000018001A82B 49 83 C2 10                             add     r10, 10h
.text:000000018001A82F 4D 8B 0A                                mov     r9, [r10+(PEB_LDR_DATA.InLoadOrderModuleList.Flink-10h)]
.text:000000018001A832
.text:000000018001A832                                 loc_18001A832:                          ; CODE XREF: mw_prepare_indirect_syscalls+5B↓j
.text:000000018001A832                                                                         ; mw_prepare_indirect_syscalls+65↓j
.text:000000018001A832                                                                         ; mw_prepare_indirect_syscalls+75↓j
.text:000000018001A832                                                                         ; mw_prepare_indirect_syscalls+83↓j
.text:000000018001A832                                                                         ; mw_prepare_indirect_syscalls+92↓j
.text:000000018001A832 4D 3B CA                                cmp     r9, r10
.text:000000018001A835 0F 84 A9 01 00 00                       jz      loc_18001A9E4
.text:000000018001A83B 49 8B 79 30                             mov     rdi, [r9+LDR_DATA_TABLE_ENTRY.DllBase]
.text:000000018001A83F 4D 8B 09                                mov     r9, [r9]
.text:000000018001A842 48 63 47 3C                             movsxd  rax, [rdi+IMAGE_DOS_HEADER.e_lfanew]
.text:000000018001A846 8B 8C 38 88 00 00 00                    mov     ecx, [rax+rdi+IMAGE_NT_HEADERS64.OptionalHeader.DataDirectory.VirtualAddress]
.text:000000018001A84D 85 C9                                   test    ecx, ecx
.text:000000018001A84F 74 E1                                   jz      short loc_18001A832
.text:000000018001A851 48 8D 34 0F                             lea     rsi, [rdi+rcx]
.text:000000018001A855 83 7E 18 00                             cmp     [rsi+IMAGE_EXPORT_DIRECTORY.NumberOfNames], 0
.text:000000018001A859 74 D7                                   jz      short loc_18001A832
.text:000000018001A85B 8B 4E 0C                                mov     ecx, [rsi+IMAGE_EXPORT_DIRECTORY.Name]
.text:000000018001A85E 8B 04 39                                mov     eax, [rcx+rdi]  ; first 4 bytes of dll name
.text:000000018001A861 41 0B C0                                or      eax, r8d
.text:000000018001A864 3D 6E 74 64 6C                          cmp     eax, 'ldtn'
.text:000000018001A869 75 C7                                   jnz     short loc_18001A832
.text:000000018001A86B 8B 44 39 04                             mov     eax, [rcx+rdi+4]
.text:000000018001A86F 41 0B C0                                or      eax, r8d
.text:000000018001A872 3D 6C 2E 64 6C                          cmp     eax, 'ld.l'
.text:000000018001A877 75 B9                                   jnz     short loc_18001A832
.text:000000018001A879 0F B7 44 39 08                          movzx   eax, word ptr [rcx+rdi+8]
.text:000000018001A87E 66 83 C8 20                             or      ax, 20h
.text:000000018001A882 66 83 F8 6C                             cmp     ax, 'l'
.text:000000018001A886 75 AA                                   jnz     short loc_18001A832
.text:000000018001A888 8B 5E 1C                                mov     ebx, [rsi+IMAGE_EXPORT_DIRECTORY.AddressOfFunctions]
.text:000000018001A88B 8B 6E 20                                mov     ebp, [rsi+IMAGE_EXPORT_DIRECTORY.AddressOfNames]
.text:000000018001A88E 8B 46 24                                mov     eax, [rsi+IMAGE_EXPORT_DIRECTORY.AddressOfNameOrdinals]
.text:000000018001A891 44 8B C2                                mov     r8d, edx
.text:000000018001A894 48 03 C7                                add     rax, rdi
.text:000000018001A897 48 03 DF                                add     rbx, rdi
.text:000000018001A89A 48 03 EF                                add     rbp, rdi
.text:000000018001A89D 49 8B CD                                mov     rcx, r13        ; void *
.text:000000018001A8A0 49 C1 E0 04                             shl     r8, 4           ; Size
.text:000000018001A8A4 33 D2                                   xor     edx, edx        ; Val
.text:000000018001A8A6 48 89 5C 24 20                          mov     [rsp+68h+AddressOfFunctions], rbx
.text:000000018001A8AB 48 89 84 24 88 00 00 00                 mov     [rsp+68h+arg_18], rax
.text:000000018001A8B3 48 89 6C 24 28                          mov     [rsp+68h+var_40], rbp
.text:000000018001A8B8 45 33 FF                                xor     r15d, r15d
.text:000000018001A8BB E8 B0 2B 00 00                          call    memset
```

Preparing system calls

The first part of it is run of the mill PEB walking and PE parsing to get names of exported functions. Note also that there is a check of `IMAGE_EXPORT_DIRECTORY.Name` against `ntdll.dll` very slightly obfuscated (it's just written backwards and split over three `cmp` instructions). This tells us the author is only interested in ntdll. That makes sense, considering they're after syscalls. There is a `memset` , to which we'll come back later.

The next block of code will check the function name for the prefixes Ki and Zw.If either prefix matches there is a call to the hashing function, which is a `ROR 8 ADD` algorithm that iterates over each `word` and uses `0x52964EE9` as a hardcoded XOR key.

```
.text:00000018001A8CD
.text:00000018001A8CD 42 8B 6C B5 00
.text:00000018001A8D2 48 03 EF
.text:00000018001A8D5 0F 84 92 00 00 00
.text:00000018001A8DB B8 48 69 00 00
.text:00000018001A8E0 66 39 45 00
.text:00000018001A8E4 75 32
.text:00000018001A8E6 48 8B CD
.text:00000018001A8E9 E8 DA FE FF FF
.text:00000018001A8EE 3D 99 44 CD 8D
.text:00000018001A8F3 75 23
.text:00000018001A8F5 48 8B 84 24 88 00 00 00
.text:00000018001A8FD 42 0F B7 04 70
.text:00000018001A902 8B 0C 83
.text:00000018001A905 48 03 CF
.text:00000018001A908 E8 7F FE FF FF
.text:00000018001A90D 48 8B 8C 24 80 00 00 00
.text:00000018001A915 48 89 01
.text:00000018001A918
.text:00000018001A918
.text:00000018001A918
.text:00000018001A918 B8 5A 77 00 00
.text:00000018001A91D 66 39 45 00
.text:00000018001A921 75 4A
.text:00000018001A923 41 8B DF
.text:00000018001A926 48 8B CD
.text:00000018001A929 48 03 DB
.text:00000018001A92C E8 97 FE FF FF
.text:00000018001A931 4C 8B 84 24 88 00 00 00
.text:00000018001A939 48 8B 54 24 20
.text:00000018001A93E 41 89 44 DD 00
.text:00000018001A943 43 0F B7 04 70
.text:00000018001A948 8B 0C 82
.text:00000018001A948 41 FF C7
.text:00000018001A94E 41 89 4C DD 04
.text:00000018001A953 43 0F B7 04 70
.text:00000018001A958 8B 0C 82
.text:00000018001A958 48 03 CF
.text:00000018001A95E 49 89 4C DD 08
.text:00000018001A963 44 3B 7C 24 78
.text:00000018001A968 74 15
.text:00000018001A96A 48 8B DA
```

```
check_start_for_Ki_and_hash:          ; CODE XREF: mw_prepare_indirect_syscalls+185↓j
                mov     ebp, [rbp+r14*4+0]
                add     rbp, rdi
                jz      loc_18001A96D
                mov     eax, 'iK'
                cmp     [rbp+0], ax
                jnz     short check_start_for_Zw
                mov     rcx, rbp
                call    mw_ror8_add_hashing
                cmp     eax, 8DCD4499h
                jnz     short check_start_for_Zw
                mov     rax, [rsp+68h+AddressOfNameOrdinals]
                movzx   eax, word ptr [rax+r14*2]
                mov     ecx, [rbx+rax*4]
                add     rcx, rdi
                call    mw_find_syscall_codeblock
                mov     rcx, [rsp+68h+arg_10]
                mov     [rcx], rax

check_start_for_Zw:                   ; CODE XREF: mw_prepare_indirect_syscalls+F0↑j
                                      ; mw_prepare_indirect_syscalls+FF↑j
                mov     eax, 'wZ'
                cmp     [rbp+0], ax
                jnz     short loc_18001A96D
                mov     ebx, r15d
                mov     rcx, rbp
                add     rbx, rbx
                call    mw_ror8_add_hashing
                mov     r8, [rsp+68h+AddressOfNameOrdinals]
                mov     rdx, [rsp+68h+AddressOfFunctions]
                mov     [r13+rbx*8+0], eax ; move hash value to indirect syscall struct
                movzx   eax, word ptr [r8+r14*2]
                mov     ecx, [rdx+rax*4]
                inc     r15d
                mov     [r13+rbx*8+4], ecx ; move address of function from ntdll to indirect syscall struct
                movzx   eax, word ptr [r8+r14*2]
                mov     ecx, [rdx+rax*4]
                add     rcx, rdi
                mov     [r13+rbx*8+8], rcx ; move ptr to syscall block to indirect syscall struct
                cmp     r15d, [rsp+68h+arg_8]
                jz      short loc_18001A97F
                mov     rbx, rdx
```

```
.text:00000018001A7C8
.text:00000018001A7C8
.text:00000018001A7C8
.text:00000018001A7C8 45 33 C9
.text:00000018001A7CB 4C 8B C1
.text:00000018001A7CE B8 E9 4E 96 52
.text:00000018001A7D3 44 38 09
.text:00000018001A7D6 74 1A
.text:00000018001A7D8
.text:00000018001A7D8
.text:00000018001A7D8 0F B7 09
.text:00000018001A7DB 8B D0
.text:00000018001A7DD 41 FF C1
.text:00000018001A7E0 C1 CA 08
.text:00000018001A7E3 03 D1
.text:00000018001A7E5 41 8B C9
.text:00000018001A7E8 49 03 C8
.text:00000018001A7EB 33 C2
.text:00000018001A7ED 80 39 00
.text:00000018001A7F0 75 E6
.text:00000018001A7F2
.text:00000018001A7F2
.text:00000018001A7F2 F3 C3
.text:00000018001A7F2
.text:00000018001A7F2
```

```
; __int64 __fastcall mw_ror8_add_hashing(_BYTE *)
mw_ror8_add_hashing proc near         ; CODE XREF: mw_prepare_indirect_syscalls+F5↓p
                                      ; mw_prepare_indirect_syscalls+138↓p
                xor     r9d, r9d
                mov     r8, rcx
                mov     eax, 52964EE9h
                cmp     [rcx], r9b
                jz      short locret_18001A7F2

loc_18001A7D8:                        ; CODE XREF: mw_ror8_add_hashing+28↓j
                movzx   ecx, word ptr [rcx]
                mov     edx, eax
                inc     r9d
                ror     edx, 8
                add     edx, ecx
                mov     ecx, r9d
                add     rcx, r8
                xor     eax, edx
                cmp     byte ptr [rcx], 0
                jnz     short loc_18001A7D8

locret_18001A7F2:                     ; CODE XREF: mw_ror8_add_hashing+E↑j
                rep retn
mw_ror8_add_hashing endp
```

A function starting with Ki will only be used if its hash matches `0x8DCD4499`; on a 22H2 version of Windows 10 I couldn't find an export from ntdll that matched such value. This routine then will act on at most one function starting with Ki and all starting with Zw. Appropriate values will populate a structure whose address was supplied to `mw_prepare_indirect_syscalls` - I've decided to call it `syscalls_organized_by_hash`. It is described below.

```
 struct syscalls_organized_by_hash {
DWORD function_hash;
DWORD ntdll_address_of_function;
QWORD ptr_to_function_syscall_block;
};
```

`function_hash` is the calculated hash for the exported function; `ntdll address of function` is an address to the function's code as pointed to by `IMAGE_EXPORT_DIRECTORY.AddressOfFunctions`; `ptr_to_function_syscall_block` is a pointer to the system call gadget related to said function, which resides in ntdll.dll memory. Remember the `memset` call earlier? It's used to zero that structure out. The `r13` register points

to it, and the additions at each address confirm the size of each struct member. After all the Zw prefixed functions are placed in the structure, an algorithm will sort their positions according to the `ntdll_address_of_function`, from lowest to highest. After this is done, the struct will contain the hashes, addresses of functions in the ntdll executable and pointers to the syscall gadgets for all functions with a Zw prefix, sorted in ascending order according to the `ntdll_address_of_function` values.

Setting up the syscalls structure

Going back to the function that resembled import by hash with what we've learned, we can see the that `mw_get_indirect_syscalls_by_hash` is supplied the `syscalls_organized_by_hash`, alongside the hash and a pointer to those empty qwords. After using the hashing algorithm to generate enums from ntdll exports, we can solve the hashes to see which APIs they intended to get the syscall code blocks to.

```
.text:00000001B001B700 E8 EF F0 FF FF
.text:00000001B001B705 48 8B 9D 80 1E 00 00
.text:00000001B001B70C 48 8D 05 8D 48 03 00
.text:00000001B001B713 48 8D 4C 24 30
.text:00000001B001B718 4C 8B C3
.text:00000001B001B71B 41 B9 69 7A 2B B1
.text:00000001B001B721 8B D7
.text:00000001B001B723 48 89 44 24 20
.text:00000001B001B728 E8 0F F0 FF FF
.text:00000001B001B72D 48 8D 05 84 48 03 00
.text:00000001B001B734 48 8D 4C 24 30
.text:00000001B001B739 41 B9 8B CF 08 C5
.text:00000001B001B73F 4C 8B C3
.text:00000001B001B742 8B D7
.text:00000001B001B744 48 89 44 24 20
.text:00000001B001B749 E8 EE EF FF FF
.text:00000001B001B74E 48 8D 05 7B 48 03 00
.text:00000001B001B755 48 8D 4C 24 30
.text:00000001B001B75A 41 B9 23 21 AF 35
.text:00000001B001B760 4C 8B C3
.text:00000001B001B763 8B D7
.text:00000001B001B765 48 89 44 24 20
.text:00000001B001B76A E8 CD EF FF FF
.text:00000001B001B76F 48 8D 05 72 48 03 00
.text:00000001B001B776 48 8D 4C 24 30
.text:00000001B001B77B 41 B9 0B B9 9B B5
.text:00000001B001B781 4C 8B C3
.text:00000001B001B784 8B D7
.text:00000001B001B786 48 89 44 24 20
.text:00000001B001B78B E8 AC EF FF FF
.text:00000001B001B790 48 8D 05 69 48 03 00
.text:00000001B001B797 48 8D 4C 24 30
.text:00000001B001B79C 41 B9 75 F0 5E 76
.text:00000001B001B7A2 4C 8B C3
.text:00000001B001B7A5 8B D7
.text:00000001B001B7A7 48 89 44 24 20
.text:00000001B001B7AC E8 8B EF FF FF
.text:00000001B001B7B1 48 8D 05 60 48 03 00
.text:00000001B001B7B8 48 8D 4C 24 30
.text:00000001B001B7BD 41 B9 65 E1 4E 66
.text:00000001B001B7C3 4C 8B C3
.text:00000001B001B7C6 8B D7
.text:00000001B001B7C8 48 89 44 24 20
.text:00000001B001B7CD E8 6A EF FF FF
```

```
call    mw_prepare_indirect_syscalls
mov     rbx, [rbp+1E70h+arg_0]
lea     rax, syscalls_struct
lea     rcx, [rsp+1F70h+syscalls_organized_by_hash]
mov     r8, rbx
mov     r9d, NT_ZwAllocateVirtualMemory
mov     edx, edi
mov     [rsp+1F70h+ptr_to_syscalls_struct], rax
call    mw_get_indirect_syscall_by_hash
lea     rax, syscalls_struct.ZwProtectVirtualMemory_ptr_to_syscall_block
lea     rcx, [rsp+1F70h+syscalls_organized_by_hash]
mov     r9d, NT_ZwProtectVirtualMemory
mov     r8, rbx
mov     edx, edi
mov     [rsp+1F70h+ptr_to_syscalls_struct], rax
call    mw_get_indirect_syscall_by_hash
lea     rax, syscalls_struct.ZwFreeVirtualMemory_ptr_to_syscall_block
lea     rcx, [rsp+1F70h+syscalls_organized_by_hash]
mov     r9d, NT_ZwFreeVirtualMemory
mov     r8, rbx
mov     edx, edi
mov     [rsp+1F70h+ptr_to_syscalls_struct], rax
call    mw_get_indirect_syscall_by_hash
lea     rax, syscalls_struct.ZwGetContextThread_ptr_to_syscall_block
lea     rcx, [rsp+1F70h+syscalls_organized_by_hash]
mov     r9d, NT_ZwGetContextThread
mov     r8, rbx
mov     edx, edi
mov     [rsp+1F70h+ptr_to_syscalls_struct], rax
call    mw_get_indirect_syscall_by_hash
lea     rax, syscalls_struct.ZwSetContextThread_ptr_to_syscall_block
lea     rcx, [rsp+1F70h+syscalls_organized_by_hash]
mov     r9d, NT_ZwSetContextThread
mov     r8, rbx
mov     edx, edi
mov     [rsp+1F70h+ptr_to_syscalls_struct], rax
call    mw_get_indirect_syscall_by_hash
lea     rax, syscalls_struct.ZwResumeThread_ptr_to_syscall_block
lea     rcx, [rsp+1F70h+syscalls_organized_by_hash]
mov     r9d, NT_ZwResumeThread
mov     r8, rbx
mov     edx, edi
mov     [rsp+1F70h+ptr_to_syscalls_struct], rax
call    mw_get_indirect_syscall_by_hash
```

`mw_get_indirect_syscalls_by_hash` works by looking for the supplied hash in the `syscalls_organized_by_hash` structure. Once that is found, it will retrieve the pointer to the syscall code block and call a function that validates said block - `mw_validate_syscall_codeblock`.

The way the verification works is simple. It will loop through the `syscalls_organized_by_hash` struct (they are actually organized by ascending order of `ntdll_address_of_function`, but I didn't know that back when I created the structure) until it finds the supplied hash. The functions are organized inside ntdll by ascending order of syscall codes - a function that uses code `0x1` is succeeded by one that uses code `0x2` and so forth. Because of this, once a hash is found the counter in `edi` will be equal to the syscall code. The validation function checks for the op codes of the `SYSCALL` and `RET` instructions.

```
.text:000000018001A73C
.text:000000018001A73C                                arg_0           = qword ptr  8
.text:000000018001A73C                                arg_20          = qword ptr  28h
.text:000000018001A73C
.text:000000018001A73C 85 D2                                  test    edx, edx
.text:000000018001A73E 74 4B                                  jz      short locret_18001A78B
.text:000000018001A740 48 89 5C 24 08                         mov     [rsp+arg_0], rbx
.text:000000018001A745 57                                     push    rdi
.text:000000018001A746 48 83 EC 20                            sub     rsp, 20h
.text:000000018001A74A 33 FF                                  xor     edi, edi
.text:000000018001A74C 48 8B C1                               mov     rax, rcx
.text:000000018001A74F
.text:000000018001A74F                                look_for_hash_in_syscalls_organized_by_hash_struct:
.text:000000018001A74F                                                        ; CODE XREF: mw_get_indirect_syscall_by_hash+20↓j
.text:000000018001A74F 44 39 08                               cmp     [rax], r9d      ; check supplied hash with syscalls_organized_by_hash structure
.text:000000018001A752 74 0C                                  jz      short hash_match
.text:000000018001A754 FF C7                                  inc     edi
.text:000000018001A756 48 83 C0 10                            add     rax, 10h
.text:000000018001A75A 3B FA                                  cmp     edi, edx
.text:000000018001A75C 72 F1                                  jb      short look_for_hash_in_syscalls_organized_by_hash_struct ; check supplied hash with
.text:000000018001A75E EB 21                                  jmp     short loc_18001A781
.text:000000018001A760                                ; ---------------------------------------------------------------------------
.text:000000018001A760
.text:000000018001A760                                hash_match:                             ; CODE XREF: mw_get_indirect_syscall_by_hash+16↑j
.text:000000018001A760 8B C7                                  mov     eax, edi
.text:000000018001A762 48 03 C0                               add     rax, rax
.text:000000018001A765 48 8B 5C C1 08                         mov     rbx, [rcx+rax*8+8] ; move pointer to syscall code block to rbx
.text:000000018001A76A 48 8B CB                               mov     rcx, rbx
.text:000000018001A76D E8 1A 00 00 00                         call    mw_validate_syscall_codeblock
.text:000000018001A772 48 8B 4C 24 50                         mov     rcx, [rsp+28h+arg_20]
.text:000000018001A777 48 89 19                               mov     [rcx], rbx      ; ptr to syscall code block
.text:000000018001A77A 48 89 41 08                            mov     [rcx+8], rax    ; ptr to syscall instruction
.text:000000018001A77E 89 79 10                               mov     [rcx+10h], edi  ; value of syscall code
.text:000000018001A781
.text:000000018001A781                                loc_18001A781:                          ; CODE XREF: mw_get_indirect_syscall_by_hash+22↑j
.text:000000018001A781 48 8B 5C 24 30                         mov     rbx, [rsp+28h+arg_0]
.text:000000018001A786 48 83 C4 20                            add     rsp, 20h
.text:000000018001A78A 5F                                     pop     rdi
.text:000000018001A78B
.text:000000018001A78B                                locret_18001A78B:                       ; CODE XREF: mw_get_indirect_syscall_by_hash+2↑j
.text:000000018001A78B C3                                     retn
.text:000000018001A78B                                mw_get_indirect_syscall_by_hash endp
.text:000000018001A78B
```

```
.text:000000018001A78C
.text:000000018001A78C                                mw_validate_syscall_codeblock proc near ; CODE XREF: mw_get_indirect_syscall_by_hash+31↑p
.text:000000018001A78C                                                                        ; mw_prepare_indirect_syscalls+114↓p
.text:000000018001A78C                                                                        ; DATA XREF: .pdata:0000000180053320↓o
.text:000000018001A78C
.text:000000018001A78C                                arg_8           = word ptr  10h
.text:000000018001A78C
.text:000000018001A78C 66 C7 44 24 10 0F 05                   mov     [rsp+arg_8], 50Fh
.text:000000018001A793 44 0F B7 44 24 10                      movzx   r8d, [rsp+arg_8] ; move SYSCALL op code (0F 05) to r8d
.text:000000018001A799 41 B1 C3                               mov     r9b, 0C3h       ; mov RET op code (C3) to r9b
.text:000000018001A79C 33 C0                                  xor     eax, eax
.text:000000018001A79E
.text:000000018001A79E                                look_for_syscall_codeblock:             ; CODE XREF: mw_validate_syscall_codeblock+28↓j
.text:000000018001A79E 48 63 D0                               movsxd  rdx, eax
.text:000000018001A7A1 66 44 3B 04 0A                         cmp     r8w, [rdx+rcx]  ; rcx = function address
.text:000000018001A7A6 75 07                                  jnz     short loc_18001A7AF
.text:000000018001A7A8 44 3A 4C 0A 02                         cmp     r9b, [rdx+rcx+2]
.text:000000018001A7AD 74 0A                                  jz      short syscall_codeblock_found
.text:000000018001A7AF
.text:000000018001A7AF                                loc_18001A7AF:                          ; CODE XREF: mw_validate_syscall_codeblock+1A↑j
.text:000000018001A7AF FF C0                                  inc     eax
.text:000000018001A7B1 83 F8 20                               cmp     eax, 20h ; ' '
.text:000000018001A7B4 7C E8                                  jl      short look_for_syscall_codeblock
.text:000000018001A7B6 33 C0                                  xor     eax, eax
.text:000000018001A7B8 C3                                     retn
.text:000000018001A7B9                                ; ---------------------------------------------------------------------------
.text:000000018001A7B9
.text:000000018001A7B9                                syscall_codeblock_found:                ; CODE XREF: mw_validate_syscall_codeblock+21↑j
.text:000000018001A7B9 48 98                                  cdqe
.text:000000018001A7BB 48 03 C1                               add     rax, rcx
.text:000000018001A7BE C3                                     retn
.text:000000018001A7BE                                mw_validate_syscall_codeblock endp
.text:000000018001A7BE
```

Once the desired entry is found, a new structure (which I've named `syscalls`) will receive a pointer to the syscall code block, a pointer to the `SYSCALL` instruction and the value of the syscall code. Although the code is a `dword`, I've made all members of struct qwords for convenience (that way I don't need to create a member for padding between different syscalls entries). The struct is as follows:

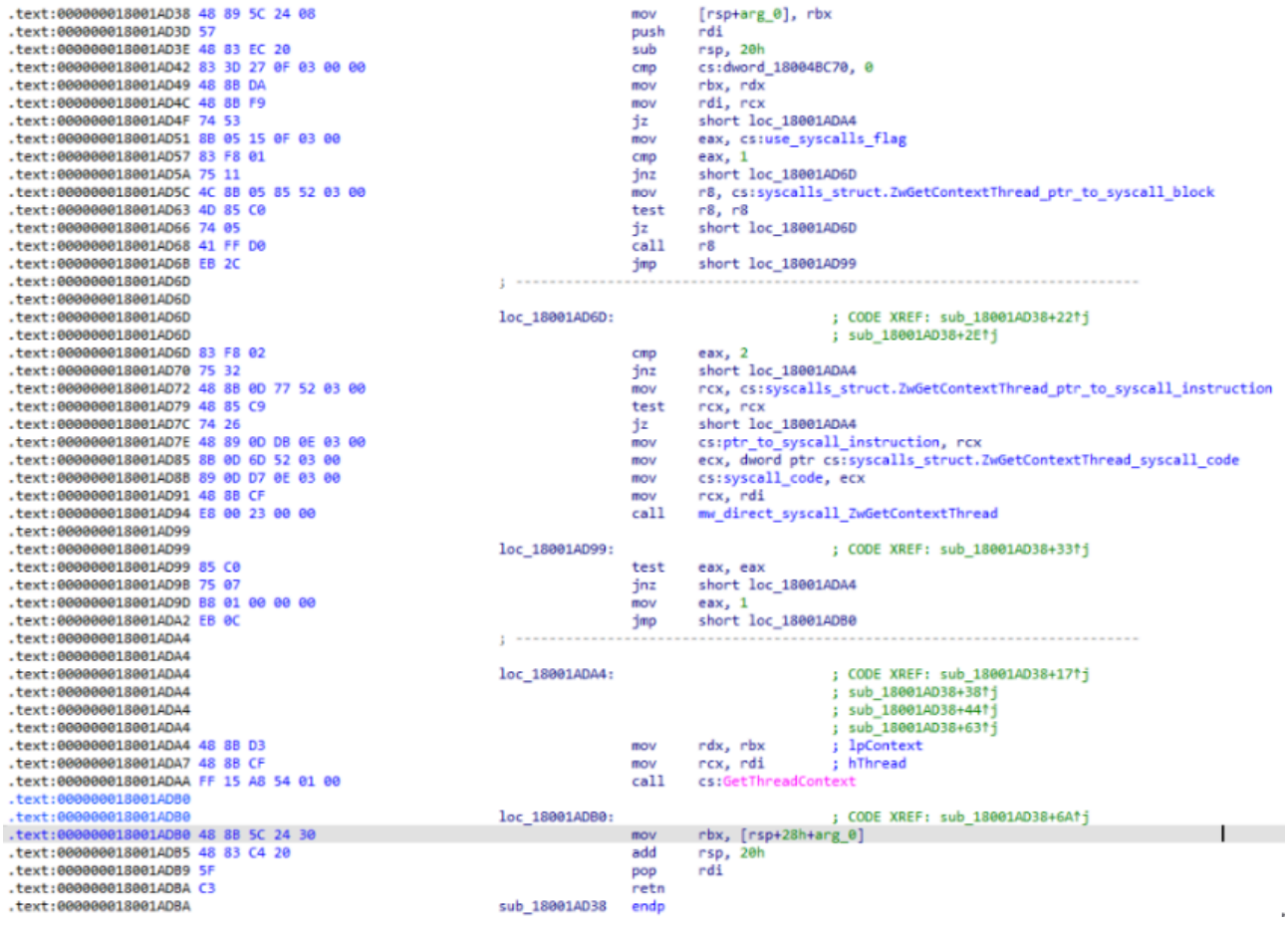
```
 struct syscalls {
QWORD ptr_to_syscall_block;
QWORD ptr_to_syscall_instruction;
QWORD syscall_code;
};
```

Now all that's left is use that model to generate the structure that will result from setting up the syscalls and apply it to the range of qwords that are passed to the `mw_get_indirect_syscalls_by_hash` function. Following cross-references to each member will lead us to places where the structure is used in the beacon code.

```
00000000
00000000 syscalls        struc ; (sizeof=0x150, align=0x8, copyof_114)
00000000                                        ; XREF: .data:syscalls_struct/r
00000000 ZwAllocateVirtualMemory_ptr_to_syscall_block dq ?
00000000                                        ; XREF: sub_18001B2B4+43/r
00000000                                        ; sub_18001B2B4+7F/r
00000008 ZwAllocateVirtualMemory_ptr_to_syscall_instruction dq ?
00000008                                        ; XREF: sub_18001B2B4+3C/r
00000008                                        ; sub_18001B2B4+78/r
00000010 ZwAllocateVirtualMemory_syscall_code dq ?
00000010                                        ; XREF: sub_18001B2B4:loc_18001B365/r
00000018 ZwProtectVirtualMemory_ptr_to_syscall_block dq ?
00000018                                        ; XREF: sub_18001B4A4+44/r
00000018                                        ; sub_18001B4A4+80/r ...
00000020 ZwProtectVirtualMemory_ptr_to_syscall_instruction dq ?
00000020                                        ; XREF: sub_18001B4A4+3D/r
00000020                                        ; sub_18001B4A4+79/r
00000028 ZwProtectVirtualMemory_syscall_code dq ?
00000028                                        ; XREF: sub_18001B4A4+B6/r
00000030 ZwFreeVirtualMemory_ptr_to_syscall_block dq ?
00000030                                        ; XREF: sub_18001B3D8+30/r
00000030                                        ; sub_18001B3D8+72/r ...
00000038 ZwFreeVirtualMemory_ptr_to_syscall_instruction dq ?
00000038                                        ; XREF: sub_18001B3D8+3F/r
00000038                                        ; sub_18001B3D8:loc_18001B458/r
00000040 ZwFreeVirtualMemory_syscall_code dq ?   ; XREF: sub_18001B3D8+8E/r
00000048 ZwGetContextThread_ptr_to_syscall_block dq ?
00000048                                        ; XREF: mw_ZwGetContextThread_wrap+24/r
00000048                                        ; mw_indirect_syscalls_setup+BF/o
00000050 ZwGetContextThread_ptr_to_syscall_instruction dq ?
00000050                                        ; XREF: mw_ZwGetContextThread_wrap+3A/r
00000058 ZwGetContextThread_syscall_code dq ?    ; XREF: mw_ZwGetContextThread_wrap+4D/r
00000060 ZwSetContextThread_ptr_to_syscall_block dq ?
00000060                                        ; XREF: sub_18001B19C+24/r
00000060                                        ; mw_indirect_syscalls_setup+E0/o
00000068 ZwSetContextThread_ptr_to_syscall_instruction dq ?
00000068                                        ; XREF: sub_18001B19C+3A/r
00000070 ZwSetContextThread_syscall_code dq ?    ; XREF: sub_18001B19C+4D/r
00000078 ZwResumeThread_ptr_to_syscall_block dq ?
00000078                                        ; XREF: sub_18001B11C+1D/r
00000078                                        ; mw_indirect_syscalls_setup+101/o
00000080 ZwResumeThread_ptr_to_syscall_instruction dq ?
00000080                                        ; XREF: sub_18001B11C+38/r
00000088 ZwResumeThread_syscall_code dq ?        ; XREF: sub_18001B11C+4B/r
00000090 ZwCreateThreadEx_ptr_to_syscall_block dq ?
00000090                                        ; XREF: sub_18001ABB8+47/r
00000090                                        ; sub_18001ABB8+81/r ...
00000098 ZwCreateThreadEx_ptr_to_syscall_instruction dq ?
00000098                                        ; XREF: sub_18001ABB8+40/r
00000098                                        ; sub_18001ABB8+7A/r
000000A0 ZwCreateThreadEx_syscall_code dq ?      ; XREF: sub_18001ABB8:loc_18001AC87/r
```

Syscall usage

Let's take a wrapper function used to get thread context as an example.



According to the value in a dword I've named `use_syscalls_flag`, the beacon will take one of three possible approaches.

- If the flag is equal to 1, it will call the desired syscall block directly; this means getting the correct code into `eax` is handled by the ntdll code.
- If the flag is equal to 2, it will call a function responsible for getting the appropriate code from `syscall.syscall_code` into `eax` and jumping to the `SYSCALL` instruction.
- If the flag is neither 1 or 2, it will simply call an API instead.

If a syscall is made by either method, the code will return 1 in `eax`. Otherwise, it returns the result from the standard API that was called. The presence of the flag leads me to think all beacons will have the mechanisms for handling syscalls. Choosing to use indirect syscalls in the builder would simply set the appropriate flag(s) in the binary, instead of producing a payload that doesn't handle syscalls at all.

Acknowledgements

Nat for suggesting looking into this in the first place and providing me with a beacon I could reverse.

Duchy for pointing out how to quickly unpack a beacon and for general help regarding structures created and used by the payload.

Source: https://github.com/dodo-sec/Malware-Analysis/blob/main/Cobalt%20Strike/Indirect%20Syscalls.md